CS 613200 Advanced Logic Synthesis Midterm Project (2021, Spring)

109062701 賴御誠

- **緒論：**

這次的專題主要是要比較 SIS 跟 ABC 兩套邏輯重寫工具的在 MCNC Benchmark 下的結果與性能表現。其中 SIS 主要早期邏輯簡化技術為主，以 SOP 與 BDD 的形式做簡化，而 ABC 則是受到近年因為 AIG 結構的特點啟發而使用該結構進行簡化。MCNC Benchmark 則是一個較 ABC 早出現的評等機制，結構較為簡單。

- **運行 SIS 的操作過程：**

先用 read_blif 讀入 blif 設計檔案，先用 print_stats 查看原始設計圖的 Literal(SOP) 數量，接著用 source 執行位於 sis_lib 底下的腳本後，再用 print_stats 查看重寫之後的 Literal(SOP) 數量，並可以用 write_blif 把結果寫入至新的 blif 檔案。

- **運行 ABC 的操作過程：**

先用 read_blif 讀入 blif 設計檔案，先用 print_stats 查看原始設計圖的 Literal(SOP) 數量，接著執行 standard scripts 腳本後，用 write_blif 把結果寫入至新的 blif 檔案。打開 SIS，再用 print_stats 查看重寫之後的 Literal(SOP) 數量。

- **比較實驗結果好壞之標準：**

使用 SIS 中 print_stats 指令查看 Literal(SOP) 數量

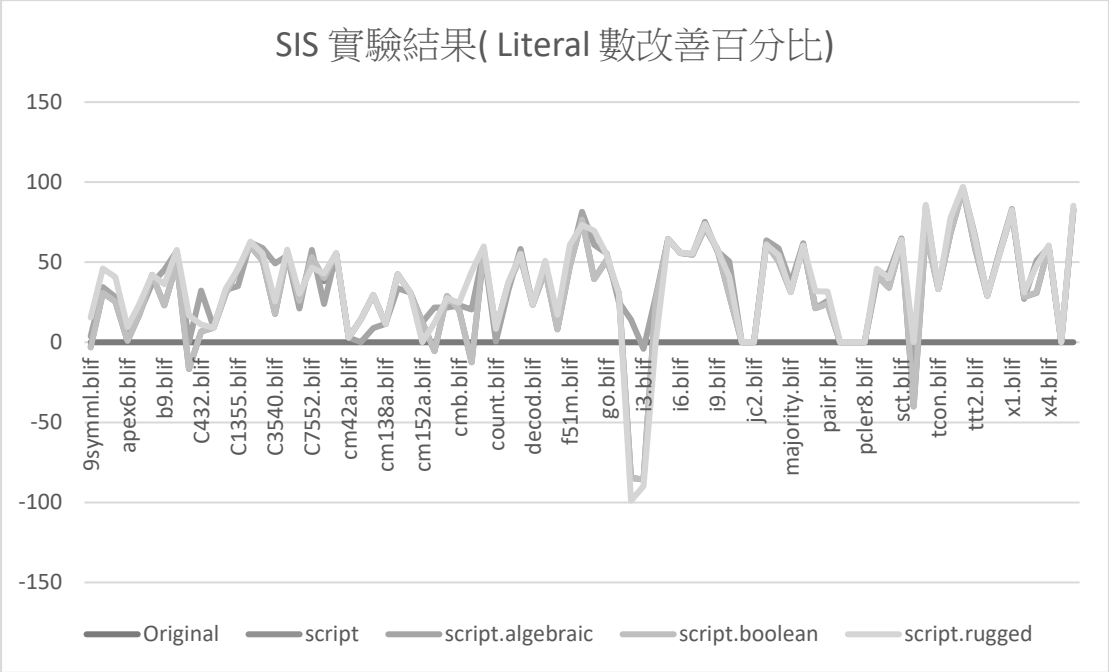
- **比較運行性能好壞之標準：**

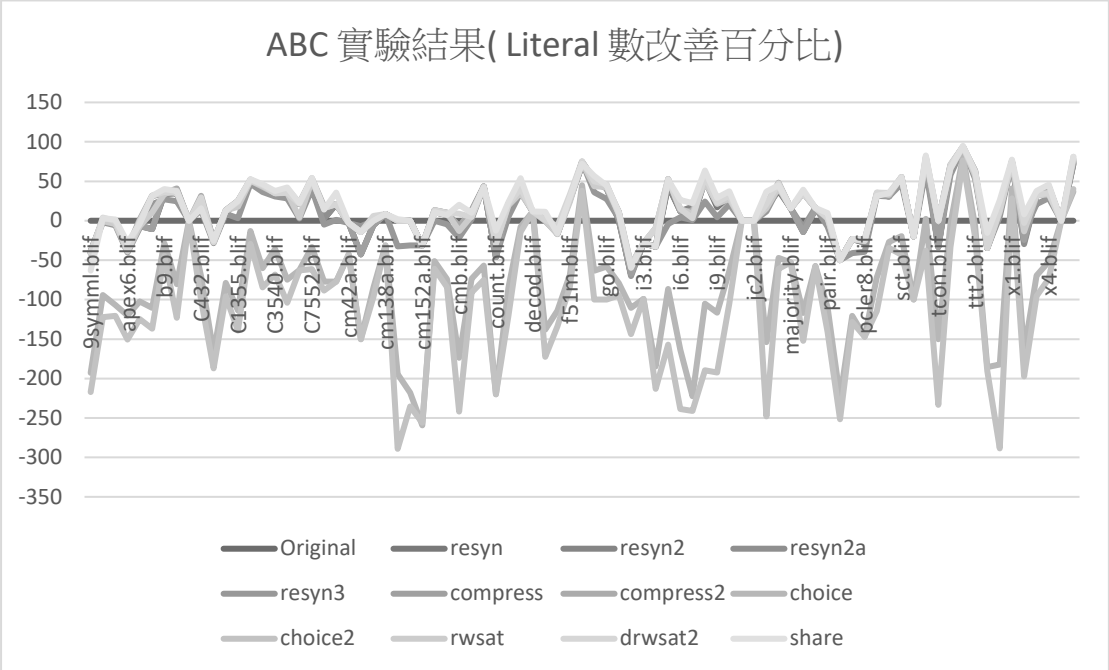
在面對特殊測資下是否會有崩潰、超時運行等情況

- **實驗進行方式：**

在 SIS 這邊，分別測試了 script、script.algebraic、script.boolean、script.rugged 四個腳本，其他的主要是因為部分指令無法使用 (比如 rlib)，抑或者是 MCNC Benchmark 的特性不能應用在某些腳本上。而在 ABC 這邊則是測試了 resyn、resyn2、resyn2a、resyn3、compress、compress2、choice、choice2、rwsat、drwsat2、share 十一個標準腳本，部分腳本因為需要輸入 AIG 結構或者無法轉換成相應的結構(比如 init)。至於測資的部分則把 MCNC Benchmark 內包含之 81 個 blif 檔案都運行一次改寫，除了 xor.blif 檔案似乎有些問題導致 SIS 讀出來都是 0 個 Literal ，同時也讓 ABC 直接程式崩潰(Segmentation Fault)外，其他測資都能順利運行(偶而有些 Warning)並正確的輸出結果。

下面的圖表為實驗結果，共計 81*(4+11) = 1215 項數據點，圖表中的每一項數字為 (原始 Literal 數 – 改寫後 Literal 數) / 原始 Literal 數 * 100%，用來代表每一種腳本對於 Literal 數的改善程度指標：





詳細實驗數據表格可以至 https://tinyurl.com/7urbusxp 查看。

- **實驗結果：**

從實驗結果來說可以發現，SIS 與 ABC 對於簡化技術的使用完全不盡相同，儘管乍看 SIS 在簡化 Literal 數量的表現上明顯好於 ABC，而 ABC 似乎在 Literal 數幾乎不減反增。不過，這並不代表 ABC 是一套不好的工具，而是因為我們計算 Literal 的方式是使用 Literal(SOP) 計數，這對於主要以 SAT 與 AIG 為主的 ABC 可能無法表現出實際的成果，另外 ABC 也還會考慮 Balancing 的問題。不過，在多數的測資來說，無論是 SIS 或者 ABC 都能減少一定數量的 Literal，達到簡化的目的。

以 SIS 四個腳本來說，script 跟 script.boolean 在簡化的結果上極為接近甚至相同，script.algebraic 則是四者最差的但仍好於原始設計，其中在 SIS 說明文件中做為最新的腳本 script.rugged 則在多個測資中都能減少最多的 Literal 數量，為四者中表現最佳。不過這些腳本在 i2.blif 與 i3.blif 中反而使輸出結果比原本更糟，尤其是腳本 script.rugged 更是四者中最差的，而原本最差的腳本 script.algebraic 則是四者中唯一有簡化 Literal 數的。

對於 ABC 的十一個腳本來說，則被分成兩個群體，choice 與 choice2 無論使用哪筆測資都是所有裡面最差的，多數時候都會大幅增加 Literal 數量，有時甚至相較原始數量直接增加 3 倍。所有裡面表現最好的為 share 腳本，在多數情況下增加最少的 Literal 數，或者減少最多 Literal 數量。其他的腳本則或多或少有相似度，但是在某些測資下又完全不像，比如 resyn、resyn2a、compress、rwsat 應該是這幾個裡面在很多的測資下都得到接近甚至完全一樣的結果。resyn2 與 resyn3 則是在最終結果上與前四者相似，但是細節比較上跟前者則沒太大關連。最後，compress2 和 drwsat2 也得到較為類似的結果，為 ABC 中 Literal 數減少量的前三名，但遠不及 share 的結果來的好。

關於運行速度的部分，SIS 與 ABC 都能在可接受的範圍內完成絕大多數測資的改寫，不過可以明顯發現 ABC 的運作速度較快，面對較大的測資也毫無壓力，而 SIS 在部分龐大測資就會花上較多的時間，而在 i4.blif 這筆測資上，script.rugged 腳本無法在有效時間內跑出結果，ABC 則沒有類似的狀況。這點與 ABC 說明文件中與 SIS 比較的部分的描述大致相同。

不過，MCNC Benchmark 中的 xor.blif 測資似乎有點問題，SIS 雖然可以正常讀入卻沒有任何 Literal 數量，而 ABC 在下了 read_blif 之後就直接崩潰退出。打開該檔案可以發現除了既有的 Input Output 之外沒有其他關於 Node 的任何描述，估計這有可能是導致這兩套軟體無法正確地讀入測資的原因。

下面的表格則是整理了 SIS 與 ABC 在 81 組測資下減少 Literal 最佳的測資數量總數，若有相同數量者則取高名次的排名：

SIS 簡化最多 Literal 數測資數量：

	原測資	script	script.algebraic	script.boolean	script.rugged
測資數	8	23	38	23	53

ABC 簡化最多 Literal 數測資數量：

	原測資	resyn	resyn2	resyn2a	resyn3	compress	compress2	choice	choice2	rwsat	drwsat2	share
測資數	21	8	13	8	8	12	24	4	4	13	23	42

- **撰寫 SIS ABC 自動化腳本：**

由於這次實驗共需要取得 1215 項數據點，換算成實際運行指令的行數為 1135(SIS 簡化查看結果) + 3522(ABC 簡化) + 1761(SIS 查看 ABC 結果) = 6418 行指令，因此勢必需要透過腳本自動化來實現。經研究 SIS 範本腳本後發現 Source 指令內容的 Script 可以再去呼叫其他現有的腳本，透過了這一個特性於是撰寫了一個可以呼叫其他簡化指令的腳本，並且能夠實現載入測資、執行簡化、查看簡化結果，並繼續載入下一組測資的功能。

此外，為了能夠讓撰寫自動化腳本的彈性修改空間更大，這邊使用了簡易的 C/C++ 程式來實現腳本批次生成，並且可以自由增減區塊指令數、總測資數、測資生成 blif 檔案之命名……等等，如此以來若需要對實驗過程做更動也只需要稍微修改腳本產生程式之程式碼即可。

相關簡易程式碼與輸出腳本範例可以至 https://tinyurl.com/42wsuvwj 查看。

- **結論：**

這次的實驗使用 81 組測資與 15 種腳本共 1215 項數據點來分析 SIS 與 ABC 兩套邏輯簡化工具的優劣之分，經過實驗後發現 SIS 在腳本 script.rugged 中能夠成功簡化最多筆測資，在 ABC 中 share 腳本則能在避免增加太多 Literal 數量下簡化與平衡最多筆測資，兩者各有優缺。在完成這次的期中計畫之後，除了對 SIS 與 ABC 兩套工具的屬性更為了解之外，也對他們背後的簡化邏輯的原理更加的清楚。